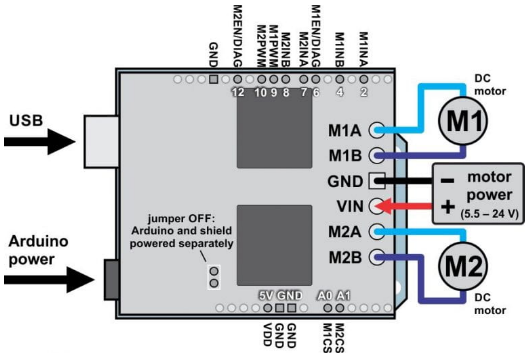
SCSE MDP Group 9

SCSE Multi Disciplinary Projects

Arduino

Architecture

<u>Motor – Pololu dual VNH5019 motor driver shield</u>



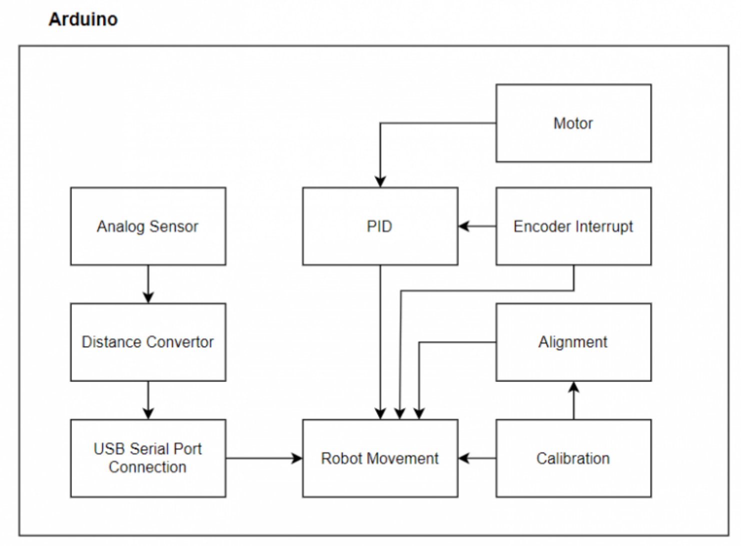
VNH5019 motor driver shied is connected to Arduino Uno to control the two bi-directional, high power DC motors. Based on the schematics, driver pins M1A, M1B, M2A and M2B are used to enable the DC motors. For this case, M1 is the Left DC Motor whereas M2 is the Right DC Motor.

<u>Sensor – SharpIR GP2Y0A21YK0F (short range) and GP2Y0A02YK0F (long range)</u>

To obtain optimal sensor data readings, we collected <u>30 samples</u> and plotted graphs to generate best-fit curve and retrieve the average calculation of one measurement. Below shows readings for every **2cm** (smaller is higher accuracy):

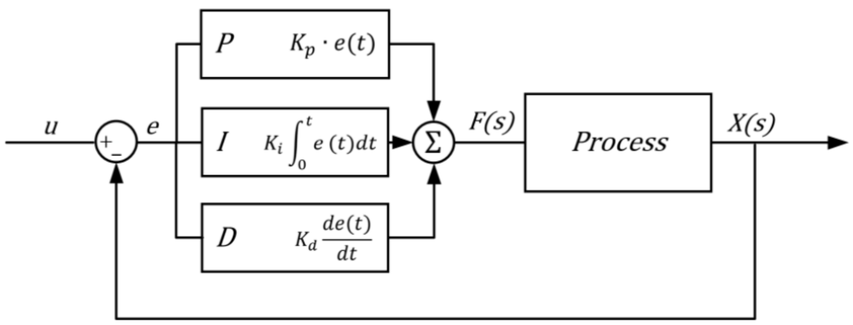
Actual Distance(cm)	PS1(FL)	PS2(FC)	PS3(RF)	PS4(FR)	PS5(RB)	PS6(LR)
2	630	626	633	623	630	586
4	614	605	554	610	582	571
6	513	503	461	513	487	554
8	435	425	396	441	415	540
10	374	366	347	383	363	517
12	332	321	309	332	322	495
14	298	289	281	298	288	474
16	270	264	256	271	266	452
18	247	244	232	250	246	427
20	228	225	216	231	227	405
22	208	210	201	212	211	385
24	195	193	188	197	195	363
26	181	181	177	184	184	343

An overview of the architecture of the Arduino System may be observed below:



Calibrations

<u>Motor Calibration</u>



By nature, M1 and M2 wheels are intrinsically different in speed due to hardware. During our runs, we discovered that M1(left wheel) produces a higher number of encoder ticks than M2(right speed). In order to get a stable motion control, a timer-based PID control interrupt system is used to synchronized the wheels. As shown in the code implementation below, we have a timer that interrupts every **80ms** to calculate the error accumulated over time, and perform calculations to match the slower wheel with the speed of faster wheel, by using *kp, ki, kd* and *grad* values*:*

```
if (timer >= 80)
{
  E2_prev_error = E2_error;
  E2_total_error += E2_error;
  E2_error = enc_m1_sum - enc_m2_sum + grad * enc_m2_sum;

  M2_speed += ((kp - offset) * E2_error) + (kd * (E2_error - E2_prev_error));
  if (modeflag == 2)
    M2_speed = (M2_speed > m2speed) ? m2speed : M2_speed;
  M2_speed += (ki * E2_total_error);

  if (modeflag == 2)
    M2_speed = max(min(m2speed, M2_speed), 0);

  motor_driver.setM2Speed(M2_speed * M2);

  enc_m1 = 0;
  enc_m2 = 0;
  timer = 0;
}
```

Since timer-based PID is voltage sensitive, we created a table with PID values for different voltage levels. A screenshot for the same is:

TURN LEFT

Voltage	Opt. KP	Opt. KD	Opt. KI	Ticks
6.31-6.34	0.5	0.1	0	380
6.40-6.45	0.4	0.1	0	370
6.50 - 6.55	0.3	0.1	0	370

TURN RIGHT

Voltage	Opt. KP	Opt. KD	Opt. KI	Ticks
6.31-6.34	0.5	0.1	0	380
6.40-6.45	0.4	0.1	0	370
6.50 - 6.55	0.3	0.1	0	370

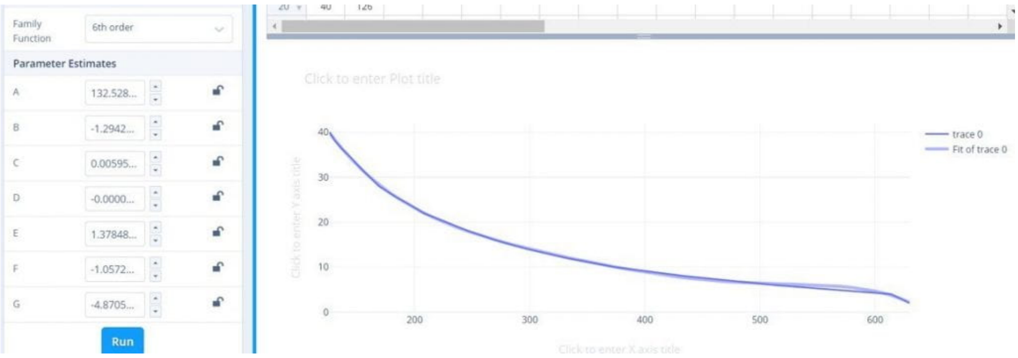
GO FORWARD/BACKWARDS

Voltage	Opt. KP	Opt. KD	Opt. KI	Ticks
6.31-6.34	4	4	1	285
6.36-6.37	4.2	3	1.5	270
6.40-6.42	4.4	3	1.5	270
6.45-6.47	4.4	3.5	1.5	265
6.52 - 6.55	4.4	3.1	1.5	258

<u>Sensor Calibration</u>

Placement: **3 short-range (front), 2 short-range (right), 1 long-range (left)**.
Steps:

1. Use a program to generate a **6th order equation** (for maximum accuracy) that best fits the plotted graph from the sensor readings data.
2. Each equation generated is unique to a sensor. This process of generating equations are repeated for all other 5 sensors.



Implementation

As mentioned earlier, the Arduino uses the timer-based PID control system, for accurate rotation of wheels, in terms of direction and movement ticks, based on the command received. For example, for forward movement, we would use the following function signature, where the parameters *M1* and *M2* guide how both wheels rotate forward, and the other 3 arguments are specific to the command:

```
PID_control(1, 1, 1.5, 0.1, 0.3); // M1, M2, kp, kd, ki
```

For exploration task, we had implemented several types of calibrations, to ensure that the robot is always positioned at the center of the grid and that its center is always at a distance of 15 cm from the walls and obstacles. The main calibration functions were *caliRightAngle()* to stay at a right angle with the wall, and *caliFrontDistance()* to stay at a distance of 15 cm from the wall. Template for code implementation:

```
void calibrate()
{
  float diff = sensorReading();

  while (abs(diff) > 0.1)
  {
    // do some small movements for adjustment
    diff = sensorReading();
  }
}
```